17 Schaltungen mit Multiplexern

In Fortsetzung von Kapitel 16 wollen wir hier eine weitere wichtige MOS-Schaltungstechnik weiter vertiefen, die in Abschnitt 11.2 kurz eingeführt wurde. Diese basiert auf der Realisierung von MOS-Logikschaltungen als Transmissionsnetze (formale Einführung in Kapitel 7). Diese Transmissionsnetze werden jedoch direkt durch Multiplexer-Schaltnetze modelliert und auch realisiert. Für ein solches Verfahren besteht Bedarf, da die später in Kapitel 18 eingeführte Methodik bei manchen Problemstellungen nur einen geringeren Optimierungs-Effekt zeigt, und für nicht seriell/parallel zerlegbare Logik nicht anwendbar ist. Außerdem gibt es Anwendungsgebiete des strukturierten VLSI-Entwurf, bei denen von vorn herein mit Multiplexern gearbeitet wird, wie beispielsweise bei konfigurierbaren Logik-Bausteinen (vgl. Kapitel 3).

Bild 17.1 zeigt die durch dieses Kapitel berührten Abstraktions-Ebenen. Zunächst werden wir im folgenden Abschnitt den Multiplexer systematischer als bisher einführen. Sodann werden wir das Prinzip des Shannon'schen Expansions-Theorems erläutern. In Abschnitt 17.1.3 wird u. a. das Prinzip des "universellen Funktionsgenerators" erläutert, der die Multiplexer-basierte Schaltungstechnik ausnutzt. In Abschnitt 17.2 wird dann gezeigt, wie die Shannon-Entwicklung über die Ableitung von Multiplexer-Bäumen zur Erstellung von Multiplexer-basierten CMOS-Schaltungen eingesetzt werden kann. Zur formelleren Beschreibung wird gelegentlich die textuelle Hardware-Beschreibungs-Sprache [3] [2] [5] *KARL-3* [4] [6] oder die graphische Sprache ABL verwendet [1] [6] (siehe auch Abschnitt 19.4).

17.1 Der Multiplexer

Als Multiplexer wird eine Schaltung bezeichnet, die n Quellen-Anschlüsse (q_0, q_1,... q_{n-1}), einen Ziel-Anschluß Z, sowie einen Adreß-Eingang SEL hat derart, daß durch einen bestimmen Wert s_i der Selektor-Variablen SEL jeweils ein bestimmter, dieser Adresse zugeordneter Quellen-Anschluß Q_i ausgewählt und zum Zielanschluß Z durchgeschaltet wird. Der Wert von SEL ist demnach also eine Art Quellen-Index. Bild 17.2 c zeigt das ABL-Diagramm eines solchen allgemeinen Multiplexers. Zur Darstellung der Zuordnung zwischen Adressen und Daten-Eingängen des Multiplexers wird im ABL-Diagramm an jeden Eingangs-Port die zugehörige Adresse geschrieben. Beispielsweise die Marke "[0]" am Eingang obersten Eingang (Bild c) gibt an, daß Signal q_0 dann nur dann an Ausgang Z durchgeschaltet wird, wenn SEL = 0 ist. Bild 17.7 stellt diese graphische Form in ABL [1] der textuellen Darstellung in einer Hardware-Beschreibungs-Sprache gegenüber (KARL-3 [3] [2]).

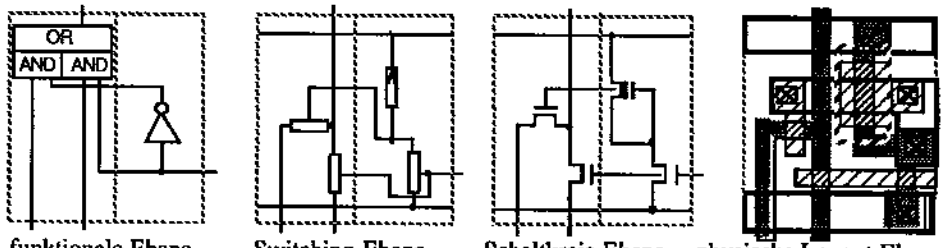

Bild 17.1: Durch dieses Kapitel berührte Abstraktions-Ebenen.

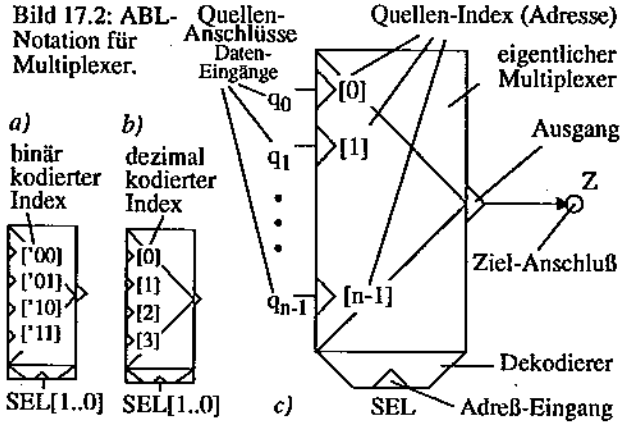

Bild 17.2: ABL-Notation für Multiplexer.

Solche Quellen-Adressen zur Markierung der Multiplexer-Eingänge können verschiedene Formate haben. Bild 17.2 veranschaulicht dies durch ABL-Diagramme eines 4-Wege-Multiplexers, wobei SEL nur zwei Bits lang ist (was durch SEL[1..0] spezifiziert ist. Die Adressen können als Bit-Vektor (Bild 17.2 a) oder als Dezimalzahl (Bild 17.2 b) angegeben werden. Dies sei an einem weiteren Beispiel mit einer 3-Bit-Adresse erläutert, womit eine von bis zu 8 Quellen selektierbar ist. Hat beispielsweise der Adreßvektor die Bit-Kombination 101, so können wir im ABL-Diagramm den zugeordneten Eingang mit ['101] als Bitvektor oder mit [5] als Dezimalzahl angeben.

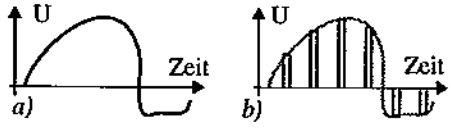

Bild 17.3: Abtastung; a) kontinuierliches Signal, b) abgetastete Form des Signales (a).

Eine historische Anwendung ist der Zeitmultiplex-Betrieb von Telefon-Leitungen. Nach Maßgabe des Abtast-Theorems [9] wird ein kontinuierliches Signal (Beispiel in Bild 17.3 a) auch in einer durch periodische Abtastung gewonnenen Pulsfolge übertragen werden (Bild 17.3 b). Der Mensch am Telefonhörer merkt den Unterschied kaum. In den Pausen zwischen den Impulsen können die Abtastpulse anderer Telefongespräche übertragen werden. So kann eine einzige Telefonleitung mehrere Gespräche "gleichzeitig" übertragen. Bild 17.6 a zeigt das Schema eines 8-fach-Zeitmultiplex. Bild 17.6 b stellt dies formal gemäß Bild 17.2 dar. Ein Multiplexer wählt hier zyklisch eine Quelle nach der anderen aus. Ein dazu synchron laufender Demultiplexer wählt zyklisch einen Empfänger-Eingang nach dem anderen aus.

Die Digitaltechnik verwendet oft Wort-Multiplexer, synthetisiert aus 1-Bit-Scheiben, wie der 4 Bit breite Multiplexer in Bild 17.5 b aus 4 Bit-Scheiben gemäß Bild 17.5 c. Auch flächensparende partielle Multiplexer wie in Bild 17.5a sind möglich, wenn die Anwendung dies erlaubt.

17.1.1 Geradeaus-Realisierung

Die Realisierung eines Multiplexers (4-Wege-Beispiel) in der logischen Ebene ist in Bild 11.5 erläutert, die des Demultiplexers in Bild Bild 17.10 a (Bild b das ABL-Diagramm, und Bild c den Schaltplan der Realisierung mit Transfer-Transistoren). Bild 11.5 veranschaulicht am Beispiel des 4-Wege-Multiplexers (Bild a) die Grundlagen aus der Sicht des Logik-Entwurfes. Bild 17.4 zeigt hierzu, daß ein vollständig dekodierter Multiplexer (a) aus zwei Teilen besteht (b) einen Dekodierer und den eigentlichen Multiplexer. Bild 11.5 b zeigt das Teil-Schaltnetz des Multiplexer-Teils allein und Bild c das vollständige Schaltnetz mit streng getrennten Teilschaltnetzen. Bild d zeigt die aus Bild c durch Logik-Minimierung gewonnene Version: Dekodierer und Multiplex-Teil sind verschmolzen. Der Demultiplexer ist ähnlich aufgebaut und kann aus dem Multiplexer abgeleitet werden wie folgt. Alle Quellen-Eingänge werden zu einem einzigen Eingang miteinander verbunden. Das Oder-Gatter am Ausgang entfällt, sodaß der Modul nunmehr n Ausgänge hat (statt eines einzigen Ausganges).

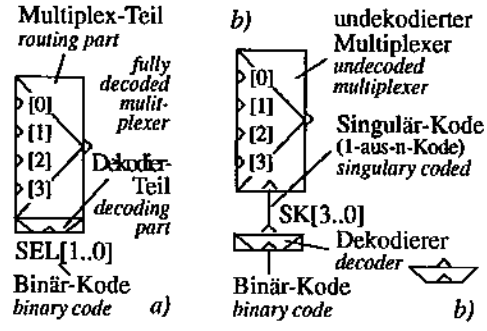

Bild 17.4: ABL-Diagramme für Multiplexer; a) vollständig dekodierter Multiplexer, b) zerlegt.

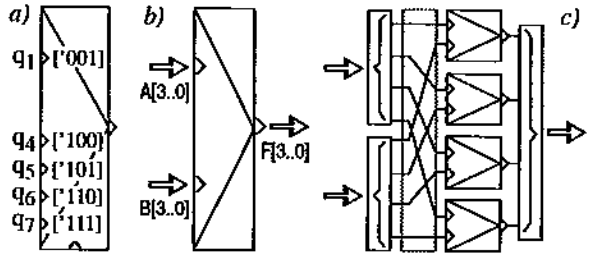

Bild 17.5: Arten v. Multiplexern: a) partieller Multiplexer, b) Wort-Multiplexer, c) Realis. von (b) d. 1-Bit-Scheiben.

17.1.2 nMOS-Realisierung mit Transfer-Transistoren

Anpaßbare Universal-Funktionsblöcke wie beispielsweise CLBs (vgl. Bild 3.4) für Anwenderprogrammierbare Logik-Bausteine (vgl. Abschnitt 3.3) werden gern durch Multiplexer realisiert (vgl. Bild 17.11). Da solche CLBs auf dem gleichen Mikro-Chip oft in sehr großer Anzahl

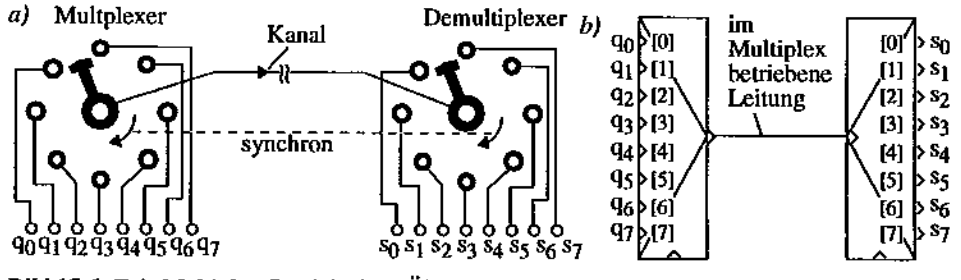

Bild 17.6: Zeit-Multiplex-Betrieb eines Übertragungs-Kanal; a) anschaulich, b) ABL-Notation.

vorkommen, ist deren Flächen-Effizienz durch minimale Transistor-Anzahl sehr wichtig. In MOS-Technologie ist der Multiplexer deshalb so interessant, weil im Vergleich zu anderen Technologien oft weniger oder gar viel weniger Transistoren benötigt werden (vgl. auch Abschnitt 11.2). Deshalb wird gern die sparsamere Realisierung durch Transfer-Transistoren bevorzugt. Bei nMOS-Technologie können für den Multiplex-Teil Transfer-Transistoren verwendet werden (vgl. Abschnitt 11.2). Bild 17.9 f zeigt den Multiplex-Teil für 4 Wege. Anstatt 17 Transistoren werden nur 4 Transistoren benötigt. Nach Minimierung unter Verschmelzung des Dekodierers mit dem Multiplex-Teil erhalten wir die Schaltung in Bild 17.9 b.

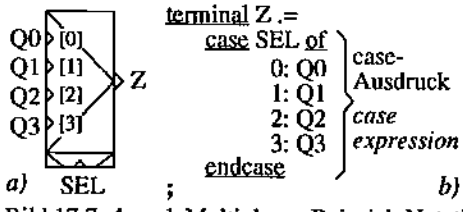

Bild 17.7: 4-zu-1-Multiplexer-Beispiel, Notationen: a) graphisch (ABL), b) textuell (KARL-3).

Der Vorteil des Transfer-Mulitplexers sei durch Zahlen verdeutlicht. Bild 11.8 a zeigt eine nMOS-Realisierung des Multiplexers nach Bild 11.5 d, wozu 25 Transistoren benötigt werden. Die Tabelle in Bild 11.9 b zeigt die Transistor-Anzahl für $2^W = 2^3 = 8$ Eingänge (55 Transistoren) und für 16 Eingänge (121 Transistoren), wobei W die Adreßwort-Länge in Bit ist. Bild 11.8 b zeigt die Realisierung dieses Multiplexers mit Hilfe von Transfer-Transistoren. Es werden (mit Dekodierer) nur noch 12 Transistoren benötigt (statt 25 Transistoren: s. oben).

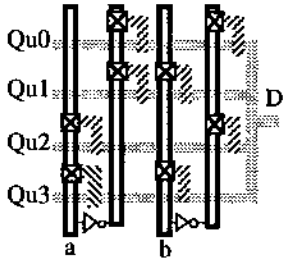

Bild 17.8: Stick-Diagramm zum Multiplexer in Bild 11.8

Bild 17.16 zeigt das Stick-Diagramm dieses in nMOS-Technik ausgeführten Multiplexers. Durch Layout-Optimierung kann der 4-Wege-Multiplexer mit erstaunlich geringem Flächenbedarf realisiert werden.

In einem Stick-Diagramm können aus Lesbarkeits-Gründen die "Sticks" nicht übereinander gezeichnet werden. Deshalb sind die Transistoren seitlich zur Eingangsleitung (Metall) gezeichnet. Beim Layout würde dies dem Schema nach Bild 17.9 a entsprechen. Ein MOS-Transistor darf jedoch auch unter einer Metallschicht liegen (dies entspricht dem Layout-Schema nach Bild 17.9 b). Probleme mit dem elektrischen Verhalten sind kaum zu erwarten, da hier die beiden übereinanderliegenden Schichten (Poly für den Gate-Anschluß des Transistors, sowie Metall) elektrisch dem gleichen Knoten angehören. Es ergibt sich eine Flächenersparnis dadurch, daß benachbarte Metall-Leitungen näher zusammengelegt werden können.

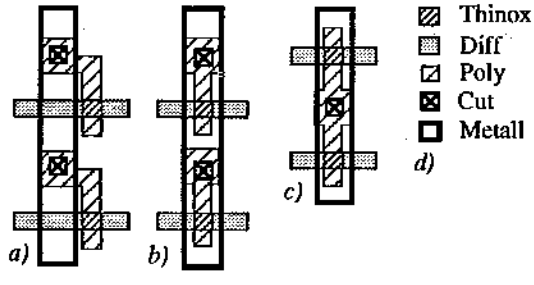

Bild 17.9: Flächensparendes Layout für die Schaltungstechnik nach Bild 17.16; a) Platzverschwendung, b) Transistorkanal unter dem Metall, c) zwei Transistoren benutzen denselben Cut, d) Legende.

Ein zusätzlicher Flächengewinn ergibt sich dann, wenn jede zweite Zeile dieser Transistor-Matrix gespiegelt wird. Dadurch können dazwischenliegende Metall-zu-Poly-Kontaktierungen je für

zwei Transistoren gemeinsam benutzt werden. Es ergibt sich ein Layout nach dem Schema in Bild 17.9 c. Es ergibt sich nach diesen Optimierungs-Maßnahmen das Layout nach Bild 11.8 c mit einem Flächenbedarf von $28 \times 29 = 812\,\lambda^2$ (gemäß Layout-Regeln nach Bild 12.8).

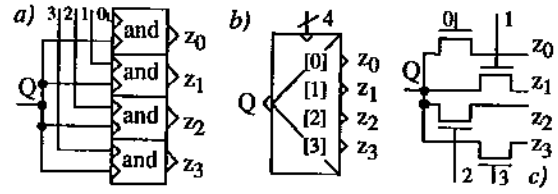

Bild 17.10: Undekodierter Demultiplexer; a) Logik-Diagramm, b) RT-Diagramm, c) Transfer-Transistor-Realis.

Die Plazierung der Transistoren in Bild 11.8 b und c entspricht eigentlich einer Matrix der Größe 4×4. Acht Positionen sind mit Transistoren belegt, die anderen 8 mit einer Drahtbrücke aus Diff-Material. Für die senkrecht durchgehenden Leitungen muß Metall verwendet werden, denn bei durchgehendem Poly würde an allen 16 Positionen ein Transistor entstehen, also auch dort, wo kein Transistor sein darf. Die Layout-Regeln für Metall verlangen 3λ Abstand und 4λ Breite (wegen der Regeln für die Durchkontaktierung). Durchgehendes Poly ist aber anwendbar [7] [8] (was allerdings nur für höchstens 4 Transistoren in einer Reihe realisierbar ist), wenn statt der Drahtbrücken selbstleitende Transistoren verwendet werden (Bild 11.8 d), denn letztere sind immer leitend: sowohl bei einer "0" als auch bei einer "1" am Transistor-Eingang. Wir erhalten nur noch einen Flächenbedarf von 399 λ^2 (siehe Layout in Bild 11.8 e),

17.1.3 ULMs

Eine weit verbreitete Anwendung des Multiplexers nach Bild 11.8 e ist die Realisierung eines universellen logischen Moduls (ULM) nach der Funktions-Tabellen-Methode (*LUT method*, LUT steht für *look-up table*). Bild 17.11 zeigt ein Beispiel für Funktionen (maximal) zweier Boole'scher Variablen. Jeweils eine Zeile (4 Bits insgesamt) aus der 4-fachen Spalte "gab" ist die LUT für eine logische Funktion. Mit diesen 4 Bits kann man 16 verschiedene LUTs darstellen, die in den Zeilen Nr. 0 bis 15 der Tabelle in Bild 17.11 angegeben und erklärt sind. In jeweils der letzten Spalte der Tabelle in

#	gab				f(A, B)	Funktion	wertig:
	g_{00}	g_{01}	g_{10}	g_{11}			
0	0	0	0	0	0	Konstante 0	0-
1	0	0	0	1	A and B	And	2-
2	0	0	1	0	B disables A	if B then 0 else A	2-
3	0	0	1	1	A	Identität A	1-
4	0	1	0	0	A disables B	if A then 0 else B	2-
5	0	1	0	1	B	Identität B	1-
6	0	1	1	0	A exor B	Antivalenz	2-
7	0	1	1	1	A or B	Or	2-
8	1	0	0	0	not(A or B)	Nor	2-
9	1	0	0	1	A coin B	Aquivalenz	2-
10	1	0	1	0	not(B)	Negation von B	1-
11	1	0	1	1	B implies A	if A then 1 else ¬B	2-
12	1	1	0	0	not(A)	Negation von A	1-
13	1	1	0	1	A implies B	if B then 1 else ¬A	2-
14	1	1	1	0	not(A and B)	Nand	2-
15	1	1	1	1	1	Konstante 1	0-

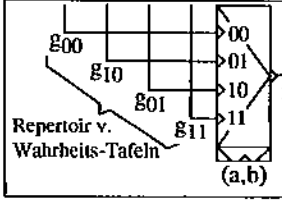

Bild 17.11: Beispiel einer ULM; univ. Funktionsgenerator für Funktionen mit 2 Variablen

Bild 17.11 ist die Wertigkeit der Funktion eingetragen. Man sieht, daß nur 10 der 16 Funktionen dyadische sind. Zwei davon sind nullwertig (die beiden Konstanten in Zeile 0 und 15) und 4 davon sind monadisch (Zeilen 1, 2, 13 und 14). Eine solche LUT kann beispielsweise aus einem Abschnitt des Operations-Kode in einem Befehlswort stammen. Enthält dieses Feld des Operations-Kode beispielsweise 0001, so liegt ein AND-Befehl vor. Ein solcher ULM kann beispielsweise für den Entwurf von Bit-Scheiben für ALUs für Mikroprozessoren verwendet werden [7] [8] (siehe Bild 20.23, wo gleich drei solcher ULM vorkommen: P, K, und R). Auch CLBs (Bild 3.4) werden oft als LUTs realisiert. Die Tabelle in Bild 17.12 gibt die Parameter von LUTs verschiedener Größen an für n allgemein, sowie n=1, n=2, und n=3. Hierbei ist n die maximale Anzahl von Variablen des jeweiligen Funktions-Vorrates. Bild 17.15 zeigt die Realisierung der gleichen Funktion f als regelmäßige Transistor-Matrix mit selbstsperrenden und selbstleitenden Transistoren.

17.2 Krause Logik aus Transmissionsnetzen

In früheren Kapiteln haben wir überwiegend solche logischen Schaltungen in MOS-Technologie behandelt, die durch Pullup- oder Pulldown-Netze realisiert wurden. In diesem Kapitel haben wir uns bisher mit reinen Multiplexer-Anwendungen befaßt. Unter bestimmten Voraussetzungen kann man aber auch krause Logik durch *Transmissionsnetze (T-Netze)* realisieren oder aus solchen ableiten. Es sei daran erinnert, daß diese Transmissionsnetze aus Transistoren bestehen, die wir global in Datenflußrichtung liegend gesehen haben und nicht quer dazu über Pullup- bzw. Pulldown-Transistoren. In Bild 11.8 b ist ein entsprechendes Transfer-Transistornetz zu sehen (vgl. auch Kapitel 7). In Bild b ist das dazugehörige Schaltbild und in Bild c das Stickdiagramm abgebildet. So können wir beispielsweise die Funktion $f = x \vee \overline{y}\, z$ durch das Transfer-Netz in Bild 17.13 a realisieren. Bei allen Eingabekombinationen, für die f wahr ist, wird die links eingespeiste Eins auf den Ausgang durchgeschaltet. Bild 17.13 b zeigt das Transistornetz und Bild c das Stick-Diagramm dazu.

Zahl der Variablen	1	2	3	n
Länge der LUT	2	4	9	n^2
Anzahl Funktionen	4	16	512	2^{n^2}

Bild 17.12: Parameter verschieden großer LUTs.

Ein Problem besteht darin, daß bei gesperrtem T-Netz der Ausgang hochohmig ist und somit f nicht erfüllt wird. Eine Lösung wäre ein Lasttransistor am Ausgang. Wir haben eigentlich kein Transfernetz mehr. Wir entwerfen ein zum ersteren inverses zweites T-Netz mit einer Null am Eingang (Bild 17.14). Immer dann, wenn f falsch ist, wird "0" zu Ausgang f durchgeschaltet.

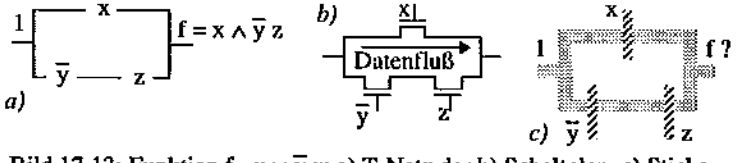

Bild 17.13: Funktion f= x ∨ ȳ z; a) T-Netz der b) Schaltplan, c) Sticks.

Bei der bisherigen Realisierung der Funktion f hatten wir 3 Selektor-Variablen (x, y, z). Am Eingang der Durchlaßpfade liegen nur die beiden Konstanten 0 und 1 an. Eine günstigere Entwurfsmöglichkeit der Schaltung ergibt sich, wenn eine oder mehrere der Variablen und ihr Komplement ebenfalls als Pfadeingänge benutzt wird (siehe Beispiel in Bild 17.17). Zum algorithmischen Entwurf solcher Schaltungen kann "Shannon's Expansions- Theorem" benutzt werden.

17.2.1 Die Anwendung von Shannon's Expansions-Theorem

Wie oben schon erwähnt, ist es unser Ziel bei der Anwendung des Satzes von Shannon, eine bzw. mehrere Eingangsvariablen und ihr Komplement an die Transferpfad-Eingänge anzulegen (Bild 17.18). Dazu müssen einige Umformungen der Funktion f durchgeführt werden.

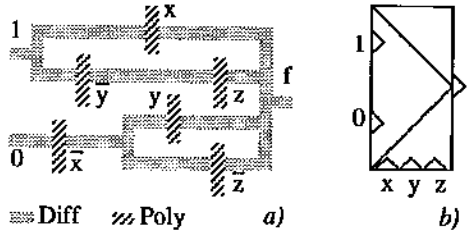

Bild 17.14: verbesserte Version der Schaltung aus Bild 17.2; a) Stick-Diagramm, b) Schema.

Diese Umformung nach dem zum Shannon'schen Expansions-Theorem beruht darauf, aus einer Funktion

$$f(x_1, \ldots x_n) \qquad (17.1)$$

die Variable x_i und deren Komplement $\bar{x}_i$ herauszuziehen. Es entstehen jeweils zwei Terme der Form

$$x_i \quad f'(x_1, \ldots x_{i-1}, x_{i+1}, \ldots x_n) \qquad (17.2)$$

und

$$\bar{x}_i \quad f''(x_1, \ldots x_{i-1}, x_{i+1}, \ldots x_n) \qquad (17.3)$$

An die Stelle i tritt statt der Variablen x_i eine Konstante 1 bzw. 0. Die beiden Terme werden nun mit OR zu einer Funktion weiter verknüpft. Wir sagen: "wir entwickeln f nach x_i". Wir erhalten

$$\begin{aligned} f = \; & x_i \quad f'(x_1, \ldots x_{i-1}, x_{i+1}, \ldots x_n) \\ \vee \; & \bar{x}_i \quad f''(x_1, \ldots x_{i-1}, x_{i+1}, \ldots x_n) \end{aligned} \qquad (17.4)$$

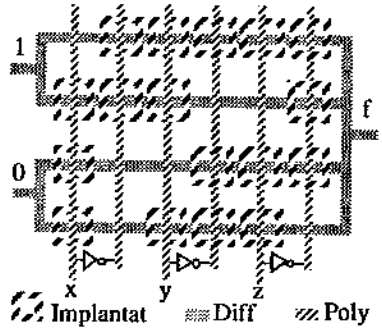

Bild 17.15: Realisierung der Funktion in Bild 17.16 mit einer regelmäßigen Transistormatrix.

Dies entspricht der Anwendung eines Multiplexers gemäß Bild 17.20 c. Eine Funktion f mit n Variablen wird aufgespalten in 1 Variable und 2 Restfunktionen mit n-1 Variablen (Bild 17.20 b). Wir stellen das Shannon'schen Expansions-Theorem etwas allgemeiner dar für den Fall, daß wir nach x_1 entwickeln.

$$\begin{aligned} & f(x_1, x_2, x_3, \ldots x_n) \\ = \; & x_1 \quad f'(1, x_2, x_3, \ldots x_n) \\ \vee \; & \bar{x}_1 \quad f''(0, x_2, x_3, \ldots x_n) \end{aligned} \qquad (17.5)$$

Wir können aber auch nach zwei Variablen entwickeln nach folgendem Beispiel (nach x_1 und x_2).

$$\begin{aligned} & f(x_1, x_2, x_3, \ldots x_n) \\ = \; & x_1 \; x_2 \quad f'(1, 1, x_3, \ldots x_n) \\ \vee \; & x_1 \; \bar{x}_2 \quad f''(1, 0, x_3, \ldots x_n) \\ \vee \; & \bar{x}_1 \; x_2 \quad f'''(0, 1, x_3, \ldots x_n) \\ \vee \; & \bar{x}_1 \; \bar{x}_2 \quad f''''(0, 0, x_3, \ldots x_n) \end{aligned} \qquad (17.6)$$

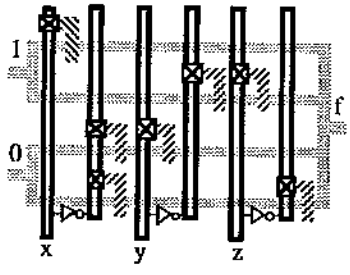

Bild 17.16: Realisierung der Funktion nach Bild 17.14 Metalleitungen für Eingänge, o. selbstleitende Transistoren.

Wir können die Funktion f aber auch nach drei Variablen entwickeln nach folgendem Schema.

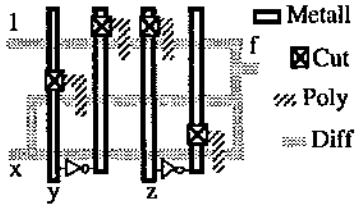

Bild 17.17: Version des Beispiel aus Bild 17.14 nach Anwendung des Shannon'schen Expansions-Theorems.

$$
\begin{aligned}
& f\ (x_1, x_2, x_3, \ldots x_n) \\
& = x_1\ x_2\ x_3 \qquad f'\ (1, 1, 1, x_4, \ldots x_n) \\
& \text{or} \quad \ldots\ \ldots\ \ldots \qquad \ldots \\
& \text{or}\ \bar{x}_1\ \bar{x}_2\ \bar{x}_3\ \ f''''''' (0, 0, 0, x_4, \ldots x_n)
\end{aligned}
\tag{17.7}
$$

Im Folgenden sei das Verfahren veranschaulicht für die Beispielfunktion nach Bild 17.14. Für unser Beispiel $f(x, y, z) = (x \vee \bar{y}\ z)$ sieht die Anwendung des Shannon'schen Expansions-Theorems wie folgt aus:

$$
\begin{aligned}
f &= (x \vee \bar{y}\ \ z) \\
&= y\ \ \wedge (x \vee 0 \wedge z) \\
&\vee\ \bar{y}\ \wedge (x \vee 1 \wedge z)
\end{aligned}
\tag{17.8}
$$

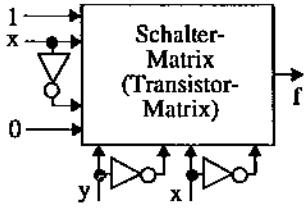

Bild 17.18: Allgemeine Struktur zur Realisierung einer Funktion mit 3 Eingangsvariablen.

Wir haben noch einen Boole'schen Ausdruck außerhalb des Funktionsblocks zu realisieren. Deshalb probieren wir mit deren Entwicklung nach zwei Variablen, nämlich den Variablen y und z, wie folgt.

$$
\begin{aligned}
&= y\ z\ (x) \quad \vee \quad y\ \bar{z}\ (x) \\
&\vee\ \bar{y}\ z\ (1) \quad \vee \quad \bar{y}\ \bar{z}\ (x)
\end{aligned}
\tag{17.9}
$$

$$
\begin{aligned}
&= \bar{y}\ z\ (1) \\
&\vee (y \vee \bar{z})\ (x)
\end{aligned}
\tag{17.10}
$$

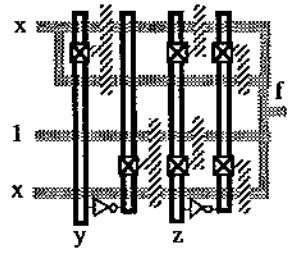

Bild 17.19: Stick-Diagramm aus Gleichung (17.9).

Wir haben nun alle logischen Verknüpfungen innerhalb des Funktionsblocks untergebracht (17.9). Diese Umformungen sowie das Ergebnis sind für dieses Beispiel in Bild 17.19 noch einmal veranschaulicht. Das Resultat läßt sich nor einmal minimieren (17.9), woraus sich das Stick-Diagramm nach Bild 17.15 ergibt.

17.3 CMOS: Multiplexer-basiert

In diesem Kapitel wird anhand eines kleinen Beispiels (Abschnitt 17.3.2) gezeigt, wie die Shannon-Entwicklung benutzt wird um Multiplexer-basierte CMOS-Schaltungen zu entwickeln. In Abschnitt 17.3.1 wird zunächst allgemein die Anwendung des Shannon-Entwicklungs-Satzes zur Entwicklung von Multiplexer-Schaltnetzen eingeführt.

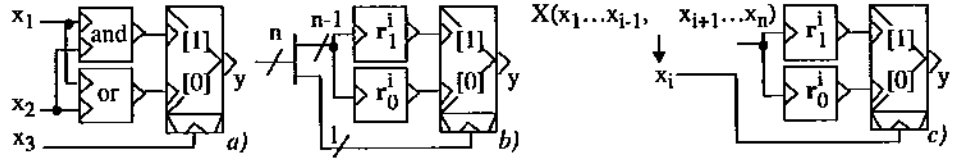

Bild 17.20: Schaltnetz mit 2:1-Multiplexer; a) Beispiel Gl. (17.11), b-c) Entwicklung nach x_i.

17.3.1 Shannon-Entwicklung und Multiplexer-Schaltnetze

Entwicklung nach einer Variablen. Man kann - wie bereits angedeutet - die Shannon'schen Formeln auch als Strukturausdrücke einer Schaltnetz-Teilrealisierung von f(x) mit Ausgangsseitigem 2:1-Multiplexer auffassen. Bild 17.20 a zeigt die Schaltnetz-Deutung für das Beispiel

$$y = \bar{x}_3(x_2x_1) \vee x_3(x_2 \vee x_1) \qquad (17.11)$$

Bild 17.20 b und c skizziert allgemeine die Entwicklung nach x_i, was die enge Beziehung zwischen Entwicklungssatz und Multiplexern aufzeigt. Es gibt formale Verfahren zur Shannon'schen Entwicklung [8]: Entwicklung nach allen Variablen, nach q von n Variablen, und nach n-1 Variablen. Man erhält Multiplexer-Bäume als Ergebnis, mit gleich großen oder unterschiedlich großen Multiplexern. Für Gleichung y = abc ∨ abc ∨ abd ∨ bcd erhalten wir beispielsweise die Bäume nach Bild 17.23.

Geschachtelte Entwicklung. Man entwickelt die gegebene Funktion nach einer ausgewählten Variablen und jede neue Restfunktion wiederum nach einer eigens gewählten Variablen. Triviale Rest-funktionen (0, 1, x_i und $\bar{x}_i$) werden nicht weiterentwickelt. Das Entwicklungs-Ergebnis beschreibt einen Baum aus 2:1-Multiplexern.

a	b	c	sum	c_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

sum:

c \ b	0	1
0	1	0
1	0	1

c_{out}:

c \ b	0	1
0	0	1
1	1	1

Bild 17.21: Addierer: Funktions-Tabellen und KV-Diagramme.

Je nach Geschick beim Wählen der Entwicklungs-Variablen entstehen Bäume mit sehr unterschiedlichen Kosten. Das Beispiel in Bild 17.25 für die Gleichung $y = \bar{a}b\bar{c} \vee ad \vee cd$ möge zeigen, wie man geschickt (Bild a) und ungeschickt entwickeln kann (Bild b). Ziel sollte sein, so bald wie möglich zu trivialen Restfunktionen zu gelangen.

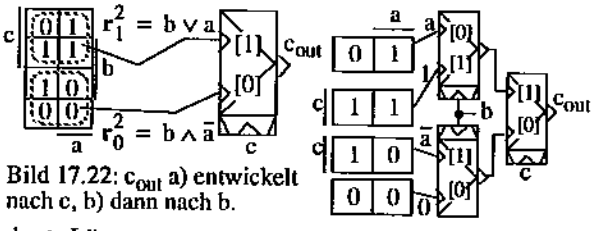

Bild 17.22: c_{out} a) entwickelt nach c, b) dann nach b.

Statt durch Probieren kann man die beste Lösung, die mit den wenigsten 2:1-Multiplexern, auch algorithmisch finden, das Verfahren beruht aber auf dynamischer Optimierung und kostet viel Zeit.

Erweiterungen. Die Konstruktionsverfahren, die wie gezeigt zu vollständigen Schaltfunktionen Multiplexer-Baumschaltnetze erzeugen, lassen sich auf weitere Aufgabenstellungen ausdehnen.

Unvollständige Schaltfunktionen. Man bricht die Entwicklung eines Zweiges ab, sobald die Restfunktion 0, 1, x_i, oder $\bar{x}_i$ lautet oder sich zu einer dieser trivialen Funktionen vervollständigen läßt.

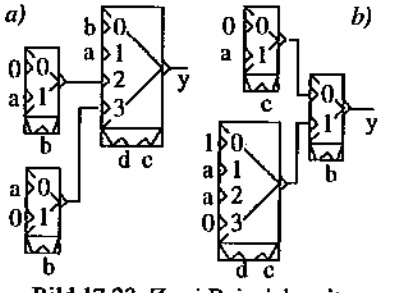

Bild 17.23: Zwei Beispiele mit verschieden großen Multiplexern.

Maschennetze. Wenn mehrfach die gleiche Restfunktion oder ihr Komplement auftritt, kann man sie einmal realisieren und ihr Teilnetz mehrfach ausnutzen. Wegen der inneren Verzweigung entsteht ein Maschennetz. Bei unvollständigen Schaltfunktionen sollte man auch prüfen, ob Restfunktionen miteinander oder mit ihrem Komplement gemeinsame Fortsetzungen haben. Allerdings kann die Realisierung der Fortsetzung teurer werden als die getrennten Teilnetze der Restfunktionen. Verschieden große Multiplexer (Beispiel siehe Bild 17.23)

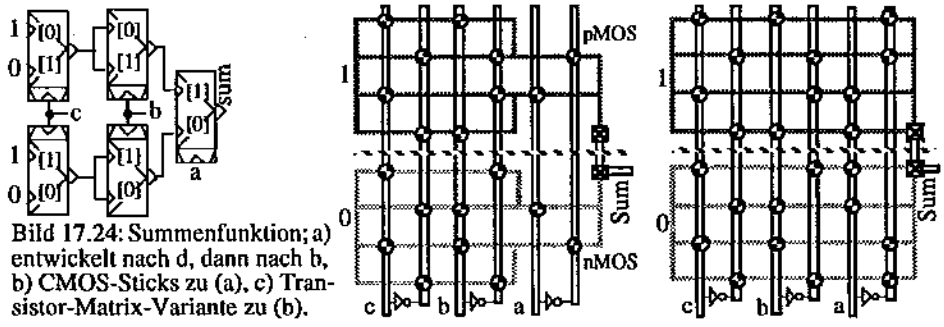

Bild 17.24: Summenfunktion; a) entwickelt nach d, dann nach b, b) CMOS-Sticks zu (a), c) Transistor-Matrix-Variante zu (b).

17.3.2 Multiplexer-basierter CMOS-Volladdierer

Dieser Abschnitt führt nun am Beispiel eines 1-Bit Volladdierers vor, wie eine Multiplexer-basierte CMOS-Schaltung entwickelt werden kann. Ähnlich dem CMOS-Transfergate, nutzen wir die gute Durchlässigkeit des p-Transistors für die logische "1" und die gute Durchlässigkeit des n-Transistors für die logische "0" aus (s. Bild 11.7 d und e). Bild 17.21 zeigt die Wertetabelle und die KV-Diagramme für die Summe s_i (auch *Sum* genannt) und den Carry-Ausgang c_{out}, die jeweils durch eigene Multiplexer-Schaltnetze realisiert werden.

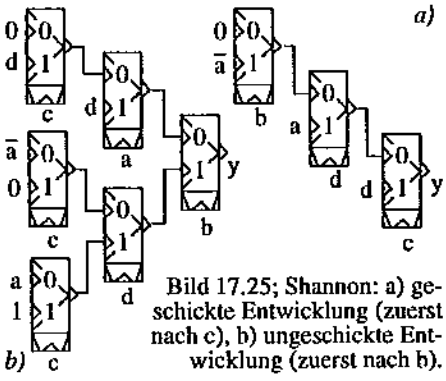

Bild 17.25; Shannon: a) geschickte Entwicklung (zuerst nach c), b) ungeschickte Entwicklung (zuerst nach b).

Das Grundprinzip ist recht einfach - wir führen Entwicklung nach allen Variablen durch. Als Ergebnis erhalten wir ein Multiplexer-Schaltnetz, an dessen Eingängen nur "1" oder "0" anliegt. Das Ergebnis der Entwicklung für die Summenfunktion *Sum* ist in Bild 17.24 a dargestellt. Bezüglich Entwicklung nach c wurden jeweils gleiche Restfunktionen nur einmal realisiert. So konnten wir uns im Schaltnetz für sum zwei Multiplexer sparen. Bild 17.24 b und c zeigen zwei Varianten der Realisierung als Pseudo-Stick-Diagramm.

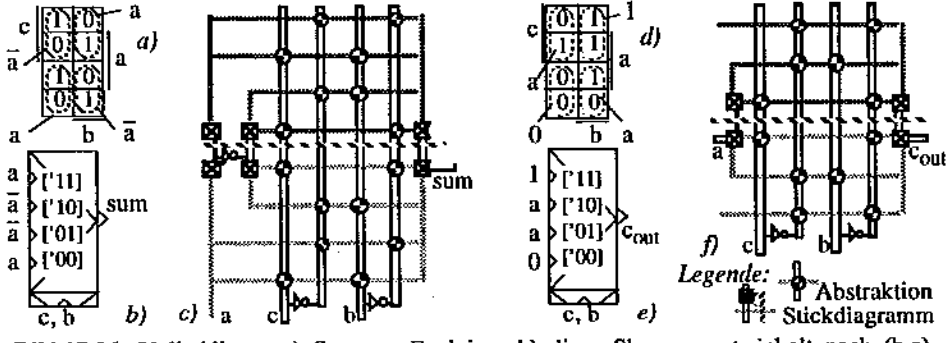

Bild 17.26: Volladdierer; s) Summen-Funktion, b) diese Shannon-entwickelt nach (b,c), CMOS-Sticks dazu, d) Übertragsfunktion, e) diese entwickelt nach (b,c), CMOS-Sticks dazu.

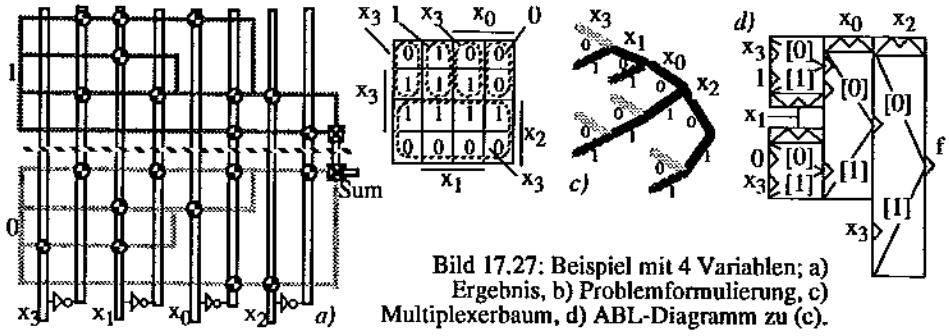

Bild 17.27: Beispiel mit 4 Variablen; a) Ergebnis, b) Problemformulierung, c) Multiplexerbaum, d) ABL-Diagramm zu (c).

Wir können aber auch nur nach zwei Variablen entwickeln: wie nach (b,c). Bild 17.26 zeigt für beide Funktionen (sum und c_{out}) die Problemformulierung (Bild a bzw. d), das Ergebnis der Entwicklung (Bild b bzw. e) und daraus abgeleitete Pseudo-Stick-Diagramme (Bild c bzw. f).

17.4 Literatur

[1] G. Girardi, R. Hartenstein, U. Welters: ABL - an interactive graphic user interface in microelectronics; in (Hrsg.: J.Encarnaçao): CAD-Schnittstellen und Datentransfer-Formate im Elektronik-Bereich; Springer-Verlag, Berlin / Heidelberg / New York 1986

[2] R. Hartenstein: Fundamentals of Structured Hardware Design: A Design Language Approach at Register Transfer Level; North Holland; Amsterdam/New York, 1977.

[3] R. Hartenstein, E. von Puttkamer: KARL - a Hardware Description Language as a part of a CAD tool for VLSI; CHDL'79, Int'l Symp. on Computer Hardware Description Languages and their Applications, Palo Alto, CA, USA, 1979; IEEE New York, 1979

[4] R. Hartenstein: KARL reference manual; CVT report, Univ. Kaiserslautern, 1986

[5] R. Hartenstein: Hardware Description Languages; North Holland, Amsterdam, 1987

[6] R. Hartenstein: KARL and ABL; in (ed.: J. P. Mermet): Fundamentals and Standards in Hardware Description Languages; Kluwer Academic Publishers, Boston, 1993

[7] C. Mead, L. Conway: Introduction to VLSI Systems; Addison-Wesley, 1980

[8] A. Mukherjee: Introduction to NMOS and CMOS VLSI System Design; Prentice Hall, Englewood Cliffs, 1986J. Newkirk, R. Mathews: The VLSI Designer's Library; Addison-Wesley, 1983

[9] H. W. Schüßler: Digitale Signalverarbeitung, Band I (Analyse diskreter Signale und Systeme); 3. Auflage, Springer-Verlag, 1992